**Problem link**- [Chocolate Pickup](https://www.codingninjas.com/codestudio/problems/ninja-and-his-friends_3125885?source=youtube&campaign=striver_dp_videos&utm_source=youtube&utm_medium=affiliate&utm_campaign=striver_dp_videos&leftPanelTab=0)

**Approach**:

Step-1 Express in terms of indices - move alice and bob together to get max chocolate.

**f (i, j1, j2)**

Step-2 explore all possible ways- 3 (alice) \* 3 (bob) = 9 directions in which we can move

Step-3 we want to obtain maximum chocolate so keep a variable maxi which will store maximum for a instance and return this value in the last.

\*\***1. recursive- solution: TC =** O(3^n \* 3^n), both alice and bob have 3 options (alice moves \* bob moves)

**//SC** = O(N) recursion stack space

**int f(int i, int j1, int j2, int m, int n, vector<vector<int>> &grid){**

**if(j1<0 || j1>=m || j2<0 || j2>=m)**

**return -1e8;**

**if(i==n-1){**

**if(j1==j2) return grid[i][j1]; //pick once**

**else**

**return grid[i][j1] + grid[i][j2]; //both can be picked**

**}**

**//explore all 9 directions**

**int maxi = -1e8;**

**for(int dj1= -1; dj1 <= +1; dj1++){**

**for(int dj2= -1; dj2 <= +1; dj2++){**

**if(j1==j2) //same col = alice or bob can pick chocolate**

**maxi = max(maxi, grid[i][j1] + f(i+1, j1+dj1, j2+dj2, m, n, grid));**

**else**

**maxi = max(maxi, grid[i][j1] + grid[i][j2] + f(i+1, j1+dj1, j2+dj2, m, n, grid));**

**}**

**}**

**//get max path for a row**

**return maxi;**

**}**

**int maximumChocolates(int r, int c, vector<vector<int>> &grid) {**

**return f(0, 0, c-1, c, r, grid); //(i, j1, j2)**

**}**

\*\***2. DP- memoization solution: //tc =** O(n\*m\*m)\*9 for all new calls

// **sc** = O(n\*m\*m) + O(n) for recursion stack space

**//memoization solution**

**int f(int i, int j1, int j2, int m, int n, vector<vector<int>> &grid, vector<vector<vector<int>>> &dp){**

**if(j1<0 || j1>=m || j2<0 || j2>=m)**

**return -1e8;**

**if(i==n-1){**

**if(j1==j2) return grid[i][j1]; //pick once**

**else**

**return grid[i][j1] + grid[i][j2]; //both can be picked**

**}**

**if(dp[i][j1][j2] != -1) return dp[i][j1][j2];**

**//explore all 9 directions**

**int maxi = -1e8;**

**for(int dj1= -1; dj1 <= +1; dj1++){**

**for(int dj2= -1; dj2 <= +1; dj2++){**

**if(j1==j2) //same col = alice or bob can pick chocolate**

**maxi = max(maxi, grid[i][j1] + f(i+1, j1+dj1, j2+dj2, m, n, grid, dp));**

**else**

**maxi = max(maxi, grid[i][j1] + grid[i][j2] + f(i+1, j1+dj1, j2+dj2, m, n, grid, dp));**

**}**

**}**

**//get max path for a row**

**return dp[i][j1][j2] = maxi;**

**}**

**int maximumChocolates(int r, int c, vector<vector<int>> &grid) {**

**//dp[r][c][c]**

**vector<vector<vector<int>>> dp(r, vector<vector<int>>(c, vector<int>(c, -1)));**

**return f(0, 0, c-1, c, r, grid, dp); //(i, j1, j2)**

**}**

\*\***3. DP- tabulation: //TC = O(n\*m\*m)\*9,**

**// SC = O(n\*m\*m)**

**//DP\_tabulation (bottom- up approach)**

**int maximumChocolates(int n, int m, vector<vector<int>> &grid) {**

**//dp[n][m][m]**

**vector<vector<vector<int>>> dp(n, vector<vector<int>>(m, vector<int>(m)));**

**//construct the base case in the dp- build last row**

**for(int j1= 0; j1<m; j1++){**

**for(int j2=0; j2<m; j2++){**

**if(j1==j2) //same value, can be obtain by alice or bob**

**dp[n-1][j1][j2] = grid[n-1][j1];**

**else //alice and bob are on different chocolates, get both**

**dp[n-1][j1][j2] = grid[n-1][j1] + grid[n-1][j2];**

**}**

**}**

**//start looping to construct entire dp from bottom to up**

**for(int i= n-2; i>=0; i--){**

**for(int j1=0; j1<m; j1++){**

**for(int j2=0; j2<m; j2++){**

**//explore all 9 directions for a position (i, j1, j2)**

**int maxi = -1e8;**

**for(int dj1= -1; dj1<=1; dj1++){**

**for(int dj2=-1; dj2<=1; dj2++){**

**int value = grid[i][j1];**

**if(j1!= j2)**

**value += grid[i][j2];**

**if(j1+dj1>=0 && j1+dj1<m && j2+dj2>=0 && j2+dj2<m)**

**value += dp[i+1][j1+dj1][j2+dj2];**

**else**

**value += -1e8;**

**maxi = max(maxi, value);**

**}**

**}**

**//store max chocolates**

**dp[i][j1][j2] = maxi;**

**}**

**}**

**}**

**//alice(0, 0) and bob(0, m-1)**

**return dp[0][0][m-1]; //i, j1, j2**

**}**

\***\*4. DP- optimized space:** //TC = O(n\*m\*m)\*9,

// SC = O(2m\*m)

**//use a nxt[m][m] to keep track of [i+1]’s j1 & j2 values and use curr[m][m] for storing current i, j1, j2 values**

**//DP\_tabulation (space- optimized)**

**//tabulation (bottom- up approach)- space optimised**

**int maximumChocolates(int n, int m, vector<vector<int>> &grid) {**

**//nxt[m][m] to keep track of (i+1)'s j1 & j2 values**

**vector<vector<int>> nxt(m, vector<int>(m));**

**//curr[m][m] to keep track of (i+1)'s j1 & j2 values**

**vector<vector<int>> curr(m, vector<int>(m));**

**//store the last row in nxt so that we can construct dp from (n-2)th row**

**for(int j1= 0; j1<m; j1++){**

**for(int j2=0; j2<m; j2++){**

**if(j1==j2) //same value, can be obtain by alice or bob**

**nxt[j1][j2] = grid[n-1][j1];**

**else //alice and bob are on different chocolates, get both**

**nxt[j1][j2] = grid[n-1][j1] + grid[n-1][j2];**

**}**

**}**

**//start looping to construct entire dp from bottom to up**

**for(int i= n-2; i>=0; i--){**

**for(int j1=0; j1<m; j1++){**

**for(int j2=0; j2<m; j2++){**

**//explore all 9 directions for a position (i, j1, j2)**

**int maxi = -1e8;**

**for(int dj1= -1; dj1<=1; dj1++){**

**for(int dj2=-1; dj2<=1; dj2++){**

**int value = grid[i][j1];**

**if(j1!= j2)**

**value += grid[i][j2];**

**if(j1+dj1>=0 && j1+dj1<m && j2+dj2>=0 && j2+dj2<m)**

**value += nxt[j1+dj1][j2+dj2];**

**else**

**value += -1e8;**

**maxi = max(maxi, value);**

**}**

**}**

**//store max chocolates**

**curr[j1][j2] = maxi;**

**}**

**}**

**//update nxt**

**nxt = curr;**

**}**

**//alice(0, 0) and bob(0, m-1)**

**return nxt[0][m-1]; //i, j1, j2**

**}**